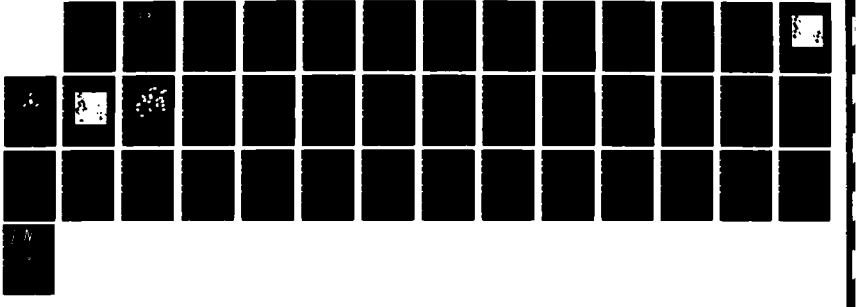
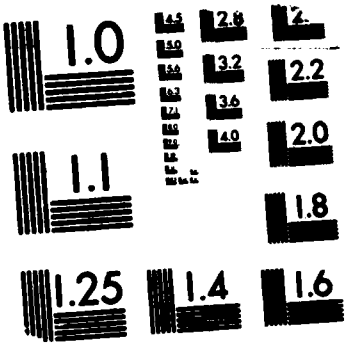
AD-A192 086 INTERMEDIATE LEVEL COMPUTER VISION PROCESSING ALGORITHM 1/1
DEVELOPMENT FOR T (U) MASSACHUSETTS UNIV AMHERST
E RISEMAN 29 NOV 87 AFOSR-TR-88-0090 F49620-86-C-0041
UNCLASSIFIED F/G 12/5 NL





DTIC FILE COPY

AD-A192 086

(2)

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

DTIC ELECTE S MAR 01 1988 D

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release, distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): aH	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 88-0090
6a. NAME OF PERFORMING ORGANIZATION: Univ of Massachusetts; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: AFOSR/NE
6c. ADDRESS (City, State, and ZIP Code): Amherst, MA 01003	7b. ADDRESS (City, State, and ZIP Code): Bldg 410, Bolling AFB, DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: SAME AS 7a; 8b. OFFICE SYMBOL (If applicable):	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-86-C-0041
8c. ADDRESS (City, State, and ZIP Code): SAME AS 7b	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. DARPA; TASK NO.; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Intermediate Level Computer Vision Processing Algorithm Development for the Content Addressable Array Parallel Processor

12. PERSONAL AUTHOR(S)
Riseman

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
R&D Status Report	FROM 01/09/87 TO 30/11/87		

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	November the author Computer

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

From September through ~~November we have~~ concentrated on the development of an Integrated Image Understanding Benchmark for vision architectures. Unlike previous vision benchmarks, this new benchmark is designed to test machine performance on a complete interpretation task. The task ~~that we have~~ chosen will require not only that common low-level vision operations be performed, but that intermediate-level operations and interactions between operations be tested as well. In fact, the emphais of the benchmark is on intermediate - or high-level processes controlling lower level processing in a top-down manner. The goal of the interpretation task is to recognize a modeled object in a cluttered environment. The input to the system is a set of models, any one of which may be present in the scene, and a pair of registered images. One of the images is from a black and white 8-bit intensity sensor, and the other is from a 32-bit floating-point range sensor. For the purposes of the benchmark, the images are artificially created.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: GILES	22b. TELEPHONE (Include Area Code): (202) 767-4933; 22c. OFFICE SYMBOL: NE

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

Intermediate Level Computer Vision Processing Algorithm Development for the Content Addressable Array Parallel Processor

University of Massachusetts: F49620-86-C-0041
Quarterly Status Report No.6

November 29, 1987

From September through November we have concentrated on the development of an Integrated Image Understanding Benchmark for vision architectures. Unlike previous vision benchmarks, this new benchmark is designed to test machine performance on a complete interpretation task. The task that we have chosen will require not only that common low-level vision operations be performed, but that intermediate-level operations and interactions between operations be tested as well. In fact, the emphasis of the benchmark is on intermediate- or high-level processes controlling lower level processing in a top-down manner.

The goal of the interpretation task is to recognize a modeled object in a cluttered environment. The input to the system is a set of models, any one of which may be present in the scene, and a pair of registered images. One of the images is from a black and white 8-bit intensity sensor, and the other is from a 32-bit floating-point range sensor. For the purposes of the benchmark, the images are artificially created.

The data in each image is designed to be insufficient to unambiguously choose the correct model object. Rather, the scenario requires the use of sensor fusion in order to complete the interpretation. Processing begins by extracting strong cues from the intensity data. These are then grouped and used to form initial model-match hypotheses. Model information is then used to probe the range data in a top-down fashion to extract portions of the modeled object for which there are weaker cues in the image data. The top-down extracted information is used to extend or veto model matches, until every object model has been either fully matched or is rejected. Because model rejection depends on the presence of strong evidence that eliminates a model from consideration, it is possible for incorrect models to match the image data, albeit poorly. Thus a match-strength value is computed for each model element and only the model with the highest average match strength is chosen as the solution to the benchmark exercise.

We have implemented a sequential version of the benchmark in order to verify the basic scenario. During the final quarter of this project we will be distributing

the benchmark to the DARPA Image Understanding Community, and will begin to implement a parallel version of the benchmark for the Image Understanding Architecture. A copy of the draft benchmark specification is attached. We feel that this benchmark will be of significant help in the design of future vision architectures.

Another project that we undertook this quarter was transportation of our VAX-based CAAPP simulator to the Sun-3/UNIX environment. That simulator is now fully integrated with the Sun windowing system and has been made essentially source-code compatible with the TI-Explorer-based simulator. The advantage of the Sun-based simulator is that it is easy to bring up at other sites on any standard Sun workstation. The Explorer-based simulator requires special hardware to be installed in the LISP machine, while the VAX-based simulator depends upon the UMass VISIONS system as a support environment

A summary of the fiscal status of the grant is:

Amount currently provided for the contract	$197,000
Expenditures and commitments to date	$165,638
Estimated date of completion of work	February 14, 1988

An Integrated Image Understanding Benchmark: Recognition of a 2 1/2 D "Mobile"

Charles Weems, Allen Hanson, Edward Riseman
Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Azriel Rosenfeld
Center for Automation Research
University of Maryland
College Park, MD 20742

Table of Contents

1. Introduction

The need for a computer vision benchmark for parallel architectures has become apparent as researchers from the fields of computer vision and computer architecture have had increasing contact over the last several years. Motion sequences at moderate resolution (512 x 512 pixels) and typical frame rate (30 frames/sec) in color (3 bytes) involves about 23.5 Mbytes of data per second. The amount of computation required for dynamic scene interpretation including the labeling of objects, surface/volume reconstruction and motion analysis is difficult to estimate; however, for many applications computational power in the range of 100 billion instructions per second, plus or minus two orders of magnitude, is probably required. Thus, vision has become a subject of major interest to computer architects.

Unfortunately, the evaluation of progress in vision architectures has been difficult. There are now quite a few interesting machines, both existing and proposed, that may be effective for at least part of the vision problem. However, computer vision transcends a wide range of representations and forms of processing. In addition, despite exciting advances in many of the subtopics of computer vision, there is currently no consensus in the research community on a unified approach to vision. There are many competing approaches and a great deal of debate has persisted. Nonetheless, it is clear that there is a need to address some of the vision/architecture issues in a form that will allow scientific insight and progress in hardware development.

Recent attempts at defining a vision benchmark include the Abingdon Cross problem, defined at the 1982 Multicomputer Workshop in Abingdon England, and the Tanque Verde benchmark suite, defined at the 1984 Multicomputer Workshop in Tucson Arizona. The most recent attempt at constructing a benchmark for vision emerged from the DARPA Image Understanding community, where a set of ten vision tasks were defined. These were: Gaussian convolution, zero crossing detection and output of border lists, connected components labelling, Hough transform, convex hull, Voronoi diagram, minimal spanning tree, visibility of vertices in a 3-D model, minimum cost path, and subgraph isomorphism.

A meeting was held in November, 1986, in Washington to compare the results of programming, simulating, or estimating the performance of a number of machines on the the individual benchmark tasks. The results [DARPA 1987 IU Workshop Proceedings pp. 298-302] were both interesting and thoroughly confusing. The data sets were only loosely specified, leading some to groups report average performance while others reported worst case performance; different groups used different algorithms; some used 32-bit floating point arithmetic while others used 16-bit integer arithmetic, etc. These results must be interpreted with extreme care.

The benchmark defined in this paper is an outgrowth of the first DARPA benchmark, and is again sponsored by DARPA. We have chosen here to address the need for an integrated vision benchmark that transcends several different representations and forms of processing that are typical of complex vision applications. The scientific gain that should result from this exercise is a better understanding of vision architecture requirements, and the performance bottlenecks in different classes of machines, so that the needs of vision processing can be better addressed in the next generation of architectures.

2. Benchmark Philosophy

In writing an integrated image understanding benchmark, the goal is to create an interpretation scenario that is an approximation of an actual image interpretation task. One must remember, however, *the benchmark scenario is not an end in itself, but rather it is a framework for testing machine performance on a variety of common vision operations and algorithms, both individually and in an integrated form that requires communication and control across algorithms and representations.* This benchmark is not intended to be a challenging vision research exercise, and in fact we believe that it *should not* be. Instead, we would like to exercise parallel architectures with a diverse range of operations that are commonly used and in a sequence that requires the sorts of transformations of data and control of processes that occur in a typical interpretation task.

In other words, the benchmark must be interesting and broad, but simple and generic.

The problem is to balance these conflicting requirements and yet define a set of tasks that effectively tests the capabilities of different architectures. A second constraint on the development of this benchmark suite is that it must use as many of the algorithms developed for the previous DARPA benchmark of non-integrated vision tasks as possible, in order to take advantage of the programming effort that has already been expended.

We have also attempted to minimize redundant operations with respect to the architectural features that are exercised. The result is that, from an image understanding perspective, the benchmark scenario may be viewed to be somewhat unrealistic and contrived as an interpretation task. However, as a benchmark, it is a valid means of testing performance.

The great variety of architectures that should be tested is itself a complicating factor in designing a benchmark. We recognize that each architecture may have its own most efficient algorithm for computing a given function. However, the tasks must be as well-defined as possible so that the results from different machines will be comparable. We should be testing the performance of the machines rather than the cleverness of their programmers. *Consequently, there should be no short-circuiting of the benchmark tasks* even if the same solution can be achieved by other, possibly simpler, means.

To this end, we are specifying a recommended method for solving each of the tasks. Whenever possible, this is the method that should be used. If the method is inappropriate for a particular architecture, then another method may be used. In such a case, the written summary of results should include a justification for the choice, and an explanation of why the recommended method was not applicable to the architecture. Note that the recommended method may not be optimal. The objective is to use a simple technique so that the results will be consistent and easier to compare. Benchmarkers are welcome to submit timings for more optimal methods in addition to those for the recommended method. However, every attempt should be made to use the recommended method, or as similar a method as possible.

We will be more fully instrumenting each task in the benchmark. In addition to simply measuring raw speed, we will develop a set of quantitative and qualitative measures to be

reported for each task, as well as for the entire scenario. Thus, it may be necessary to run the benchmark more than once in order to gather the required results. For example, if it proves difficult for a system to separate the processing time from the time required to output the instrumentation data for an intermediate result, then it may be simpler to rerun the benchmark with the instrumentation output disabled in order to determine processing times. Note that this instrumentation will be specified in a later communication.

In order to make the resulting data more meaningful, we are attempting to constrain the variability in the experiments as much as possible. The benchmark designers (the University of Massachusetts and the University of Maryland) have assumed the responsibility of testing this benchmark using traditional (sequential) methods before parallel programming and analysis begins. This has removed ambiguity and led to more precise statements of the problem, detailed specifications of the benchmark tasks, and code for the sequential benchmark and the generation of image data. The input data sets will be provided to those participating in the study so that all machines will be evaluated on the same test data, and code for generating additional test data will also be distributed. The benchmark designers have the responsibility to produce data sets that test the problem domain fairly, avoiding situations that are unrealistically complex and beyond the reasonable capability of any machine, while testing many interesting empirical situations.

In response to the results of the first DARPA benchmark exercise, we are attempting to develop guidelines for factoring in differences in hardware technology and the scalability of architectures. This will of course, require the specification of technology assumptions by each participating benchmark group.

As a final note, we realize that every benchmark exercise has a tendency to turn into a horse race. However, we hope that the result of this exercise will be the creation of a much more useful set of data and lessons than simply some lap times that are only partially interpretable. The additional constraints and required instrumentation are our attempt to ensure the usefulness of the benchmark results, and provide scientific insight and progress in an exciting as well as confusing research area.

3. Features of the Benchmark

- It involves a simple image domain with well-defined, well-behaved objects.
- It requires both bottom-up (data-directed) and top-down (knowledge or model-directed) processing. The top down processing can involve processing of low- and intermediate-level data to extract additional features from the data, or can involve control of low- and intermediate-level processes to reduce the total amount of computation required.
- It tests low-level operations such as convolution, thresholding, connected components labeling, edge tracking, median filter, Hough transform, convex hull, and corner detection.
- It requires utilization of information from two sensors in order to complete the interpretation process.
- It tests grouping operations and graph *matching*, as representative examples of intermediate-level and high-level processing, respectively.
- It requires the use of both integer and floating point representations.
- It involves the presence of occlusion (ie. absence of data) so that issues of partial matching must be considered.
- It tests the communication channels between the symbolic and numeric levels of processing.

4. Overview of the Integrated Benchmark

This benchmark task suite involves recognizing an approximately specified 2 1/2 D "mobile" sculpture composed of rectangles, given images from intensity and range sensors. It is our intention that the test images be designed so that neither, by itself, is sufficient to form a complete match.

The object to be recognized is a collection of rectangles of various sizes, brightnesses, two-dimensional orientations and depths. It can be thought of as a semi-rigid mobile consisting of suspended rectangles floating in space with fixed spatial relationships. To simplify the task, each rectangle is oriented normal to the Z axis (the viewing axis) and the image is constructed under orthographic projection. The model of the object that is provided is approximate in the sense that the sizes, orientations, and depths of the rectangles as well as their spatial relationships are constrained to within some tolerances.

The rectangles that make up the object are interspersed with additional extraneous rectangles in the scene from which the two images are taken. These additional rectangles may occlude portions of the mobile object, and some of the adjacent rectangles in the scene may have very similar brightnesses.

One of the images is from a depth sensor, and the other is from a visible light sensor (B&W). The depth image is a 512×512 array of 32-bit floating-point values. The intensity image is a 512×512 array of 8-bit integer values. Figure 1 shows an intensity image of a sample object. Figure 2 shows a depth image of the same object. Figure 3 shows the same object as in Figure 1, with extraneous rectangles added. Figure 4 shows a depth image of the same object with extraneous rectangles added. In the depth images (Figures 2 and 4), darker rectangles are closer. (Note that some of the rectangles in the depth images have been lost in the printing process.)

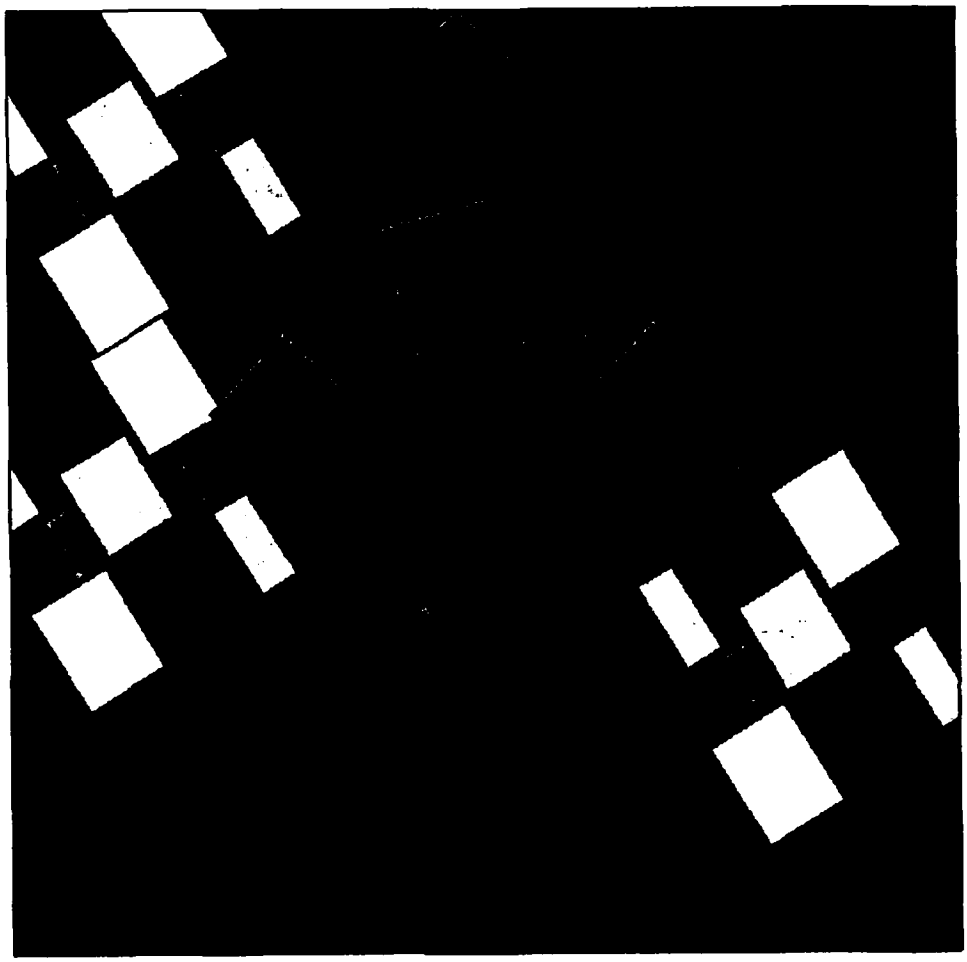


Figure 1. Intensity Image of Example Model

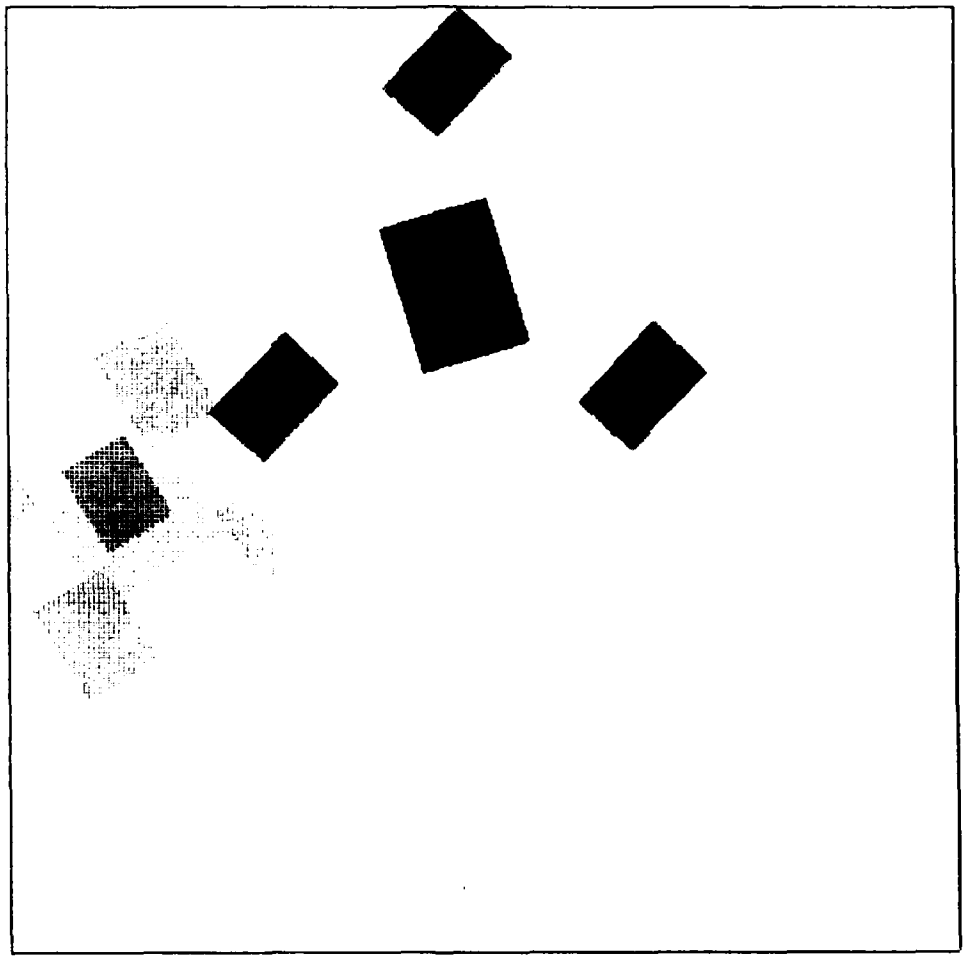


Figure 2. Depth Image of Example Model

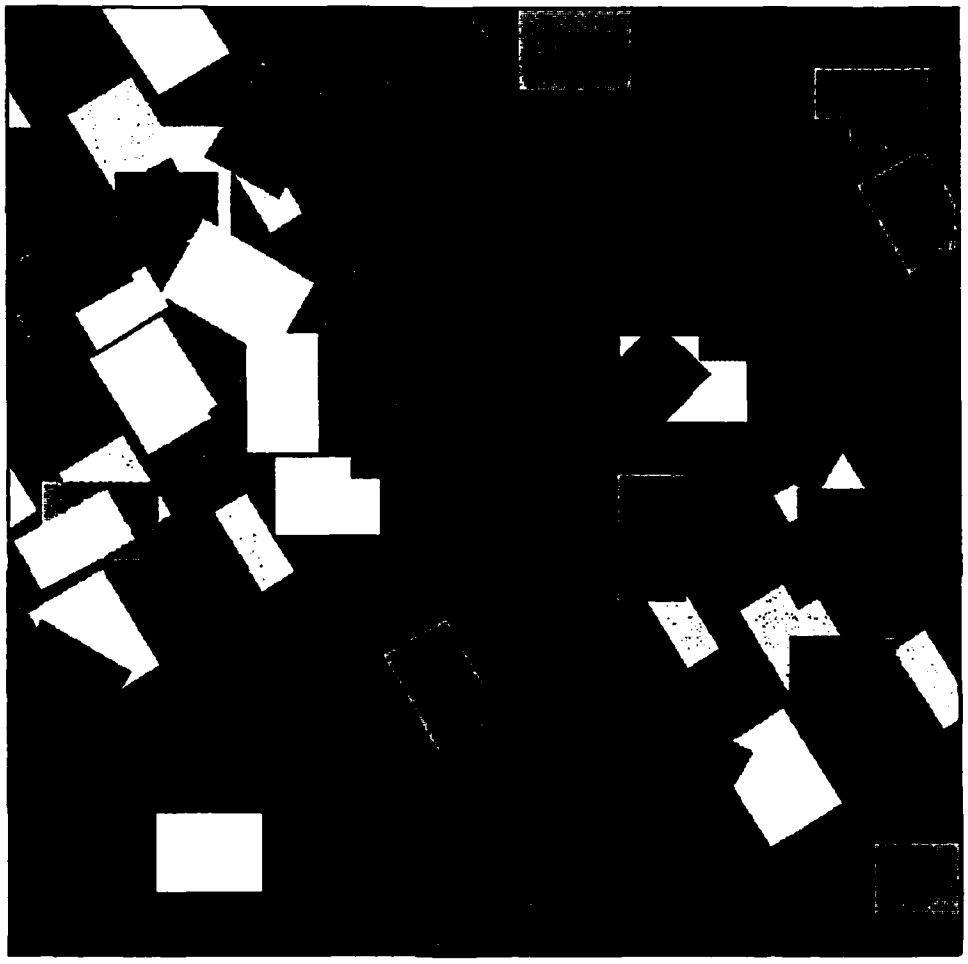


Figure 3. Example Benchmark Intensity Image

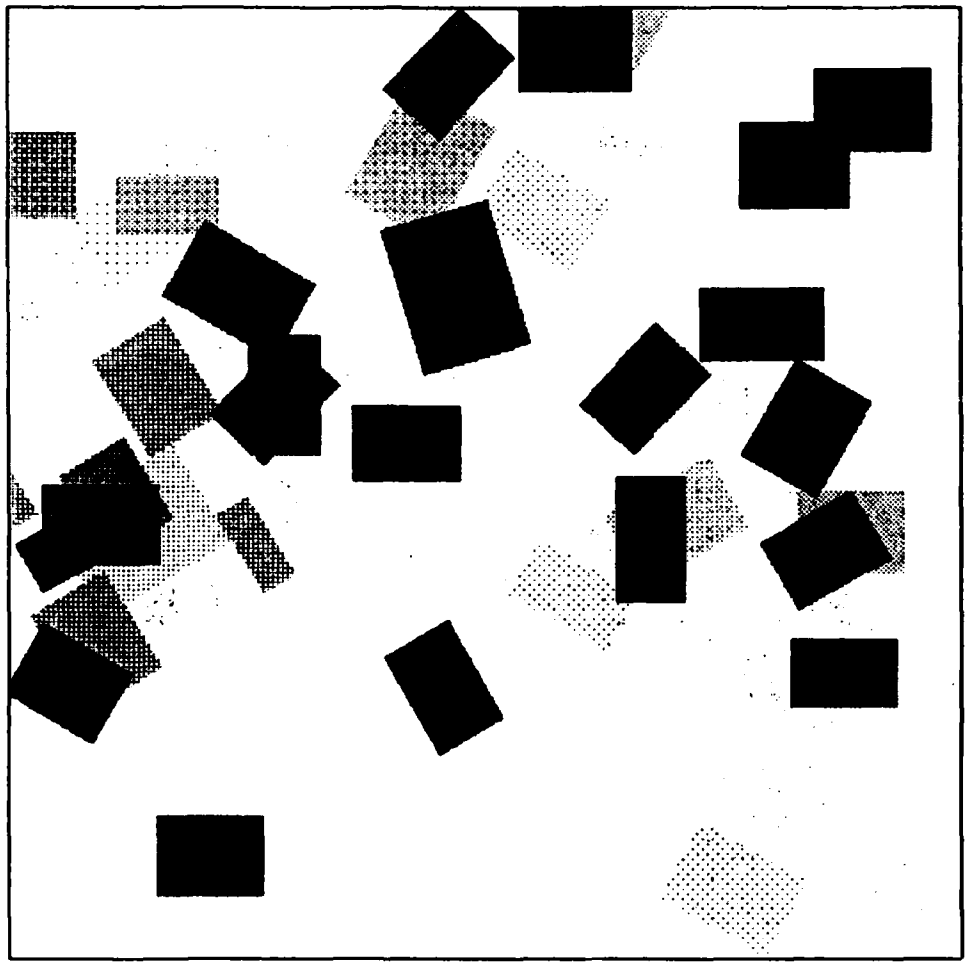


Figure 4. Example Benchmark Depth Image
(before addition of sensor noise)

A set of mobile models is provided, any one of which may be present in the images. The goal is to determine which of the models is actually present, the degree to which it is visible (matchable), and to update the model with positional data that has been extracted from the images.

5. Description of a Mobile Model

A mobile is a collection of rectangular surfaces that are oriented parallel to the image plane. This results in a constant depth for each rectangle. A mobile can be thought of as a hanging mobile looked at from directly above. The rectangles are suspended perpendicular to the force of gravity by invisible links. The rectangles can have any one of several different intensities.

A mobile model is described by a tree structure that represents the invisible links, increasing in depth, between the rectangles of the mobile. Each node of the tree contains depth, size (lengths of the major and minor axes), orientation, and intensity information for a single rectangle. The links of the tree describe spatial relationships between certain pairs of rectangles. Each model link represents the two-dimensional projection of a three-dimensional mobile link into the plane of the parent rectangle and is described by a bearing and a distance from one rectangle's center point to another rectangle's center point. Thus the spatial relationships between rectangles are specified in the X and Y dimensions by the links, and in the Z dimension by the depths of the nodes.

Each rectangle defines a new coordinate system for its child links that is relative to its own position. The center of the parent rectangle becomes the origin of the coordinate system for links emanating from it. The model's coordinate system is rotated and shifted with respect to the image coordinate system.

The model given does not exactly represent the mobile pictured. The links in the mobile can stretch and sway. This means that the length and bearing of a mobile link may be slightly different from the nominal values given in the model. Further, the sizes and depths of the recentangles are given with an associated imprecision. Intensity is not subject to any variation from the model-specified nominal values.

An instantiation of the model is created by randomly perturbing some of the parameters of the model: a) the nominal depth, size, and orientation of each rectangle, and b) the nominal angle and distance of each link given in the model. The perturbations are obtained by sampling a random Gaussian distribution (truncated at $\sigma = 3$). The scene is then

created by orthographically projecting an instantiation of the model onto the image plane.

Note that since each link is perturbed individually, the position of a rectangle connected by a single link to a known rectangle is constrained to a small area. However, rectangles that are two or more links from a known rectangle have significantly greater uncertainty in their relative position, and thus are constrained only to a much larger area.

A set of error factors (tolerances) that define the maximum perturbation of each model element's parameters will be provided with each data set. The different error factors are:

$$\begin{aligned} E_{xy} &= \text{Error in } X \text{ and } Y \text{ position (link angle and distance)} \\ E_L &= \text{Error in rectangle size (axis lengths)} \\ E_D &= \text{Error in depth} \end{aligned}$$

Note that E_{xy} specifies the error factor for both a link's angle and its distance. This is because E_{xy} is actually the distance that the child rectangle can shift in the $X-Y$ plane, relative to the nominal position specified by the link. E_{xy} may thus be thought of as the *radius of an error circle, centered at the nominal position* of the child rectangle, which defines the maximum $X-Y$ displacement of the center of the instantiated child rectangle.

Roughly ten mobile models will be provided with the test images and it is the goal of the benchmark processing to determine which model best matches the image data, and to update the positional data in that model.

6. Description of Intensity Image

The intensity image consists of an array of 512×512 8-bit pixels. The upper left pixel in the image is assumed to have the coordinates (0,0) in the discussion that follows, with the X coordinate increasing to the right and the Y coordinate increasing downward. The rectangles in the image appear against a black (level 0) background. Each rectangle will have a constant gray level, selected from a small number of levels (say 8). The intensity image is noiseless and created as if only one ray from the scene reaches every pixel: the ray perpendicular to the image plane and through the center of that pixel. Therefore, there

is no aliasing and all rectangles form sharp boundaries with any occluding or occluded rectangles of different intensity, and with the background.

In addition to the rectangles that are present in the model, the scene contains spurious rectangles that occlude, or are partially occluded by, portions of the model. All sorts of coincidental alignments of rectangles may occur. In general, it will be impossible to completely extract, via data-directed processing, all of the object model rectangles from the intensity image alone since some rectangles may be occluded (partially or completely) by others. It may also be the case that two rectangles of the same gray level are adjacent or overlap. Thus, it is possible that from 1 to 4 of the corners of any given rectangle may not be visible, thereby producing ambiguity in the matching process. Note that it will be possible to extract some of these rectangles from the depth data even though they are hidden in the intensity data. For example, a large spurious rectangle might be located behind a smaller model rectangle with the same gray level, so that they are distinguishable in the depth image, though not in the intensity image.

7. Description of Depth Image

The depth image is a 512 $\times$ 512 array of 32-bit floating point values (IEEE standard representation). A larger pixel value indicates greater depth. The coordinate system is the same as that of the intensity image, and the depth image is registered with the intensity image. The background is at a constant depth which is greater than the depth of any rectangle in the image. At first, a noiseless image is created by drawing the spurious and model rectangles in greatest to least depth order. Because the invisible links of a mobile model are similar in length, the result is a clustered distribution at several principal depths. Within each depth cluster, the depths of the individual rectangles differ by just a small amount relative to the differences in depth between clusters. The spurious rectangles are also present in the depth image and have roughly the same distribution as the model rectangles. Simulated Gaussian sensor noise is added to the noiseless depth image to produce the actual test data. The distribution of the noise is scaled and truncated so that individual pixels will have a depth error that at most causes them to appear at different

depths within a cluster but is not so great that they will appear to belong to rectangles of a different depth cluster. Thus, it is possible for a pixel in one rectangle to have a depth value that appears to make it a member of an adjacent rectangle in the same depth cluster, and therefore depth boundaries will be unreliably extracted.

The placements and depths of the spurious rectangles will be such that some of the model rectangles may be impossible to extract from the depth image. In the intensity image, however, these rectangles may be easier to extract. For example, a large spurious rectangle may be placed only slightly behind a smaller model rectangle, so that the difference in their depths is lost in the noise, but the two rectangles might differ significantly in gray level.

8. General Overview of the Processing Scenario

All of the processing steps given must be included in the parallel versions of the benchmark. Furthermore, the general scheme of the processing described at each step should be followed. Included in the description of the steps are thresholds and other parameters, which should be used as given.

Processing begins with some low-level operations on the intensity and depth images, followed by intermediate-level grouping. Given these initial results, the remainder of the benchmark task involves the following steps: Initial rectangle hypothesis generation, initial model matching, top-down depth verification for the initial match and extension of the model match for disambiguation and model update, and result presentation.

8.1 Initial Processing of the Intensity Image

The low-level operations on the intensity image consist of identifying connected components and finding the corners of each connected component based on a K-curvature operation. The initial processing of the intensity image also includes an intermediate level grouping operation that consists of creating good hypotheses for rectangles from the lists of corners around connected components in the image. The result of the initial intensity image

processing is thus a set of connected component tokens. The only feature that is extracted from each connected component as a whole is its intensity. However, each component region has associated with it a list of the corners that were extracted from its boundary.

8.2 Initial Processing of the Depth Image

The low-level operations on the depth image consist of smoothing via median filtering, computing the magnitude of the gradient, and thresholding the gradient magnitude. The result of the initial depth image processing is an image array that represents points in the depth data that have large gradient magnitudes. The smoothed depth image is also used in later stages of processing.

8.3 Intermediate-Level, High-Level, and Top-Down Processing

Intermediate level processing starts with bottom-up grouping of right-angle corners, in *component tokens*, in order to generate rectangle hypotheses. The resulting candidate rectangles form the basis of the initial model graph matching operation. The intermediate level operation on the depth image is a top-down directed search for expected rectangles, incorporating a spatially local Hough transform with model-constrained ranges on the parameters for each rectangle. The high level operations are first, constrained subgraph-to-subgraph matching to choose and orient the models to be matched; and second, top-down control of probes into the depth and intensity images to find and fix the parameters of the rectangles that are required to fill out the chosen models. As a concluding step, an image is produced that represents the single best model match as an overlay with the original intensity image.

The goal of the first graph match step is to attempt to establish the most likely positions and orientations of the modeled objects in the image and possibly to eliminate some of the models from further consideration for matching. Since only some of the model rectangles will have been extracted from the intensity image, portions of each model will have no match. The unmatched portions of a graph model are used to focus attention in the depth

image so that additional localized features can be extracted and the model can be extended through the use of context. This match extension step is further divided into three parts that are repeated for each model rectangle: model directed rectangle detection, rectangle depth and intensity verification, and model update.

9. Low-Level, Bottom-Up Processing

9.1 Intensity Image

9.1.1 Label Connected Components

Task: Each connected component in the image is given a unique integer label. A connected component is any contiguous collection of pixels that have the same intensity level value. Contiguous is defined as adjacency in any of four directions (N,S,E,W). The result is an image-size array where each pixel in a component has the value of the component label.

Recommended Method: Each pixel has an address associated with it that is the concatenation of its binary row and column number (in a 512×512 image, the resulting address is an integer with 18 significant binary digits, the column number being the low order 9 bits of the address). Each connected component takes as its label the lowest pixel address value from among the members of the component. This may be done by propagating the lowest address from the pixel at that location throughout the component. The order in which the propagation proceeds (ie. radially, by rows or columns, through distance doubling, etc.) is left to the choice of the benchmark programmer.

9.1.2 Compute K-curvature and Extract Corners

Task: For each connected component, compute the K-curvature of the boundary of the component. (The boundary is the set of pixels that are members of the component

and are adjacent (N,S,E,W) to at least one pixel that is a member of another component). The K-distance is a parameter supplied with the data set (in the test set, a distance of 4 is used). The K-curvature values are then smoothed using a one-dimensional Gaussian mask that is 7 elements wide in order to eliminate multiple local peaks near a corner. The first derivative of the smoothed curvature values is computed and zero-crossing points along the derivative of the boundary are determined. Corners are defined to be those points in the smoothed curvature values that exceed a specified threshold (ie, have high curvature) and that correspond to a zero-crossing (in order to select the point with maximum local curvature).

Recommended method: K curvature is computed for each border pixel by tracking along the border pixels of each region for a distance of K (equal to 4 in this case) in each direction. A scaled value is computed that is proportional to the angle formed by the straight lines linking the center pixel with its neighbors that are K edge points away in each direction. Relative angles between these lines are computed with a *table lookup operation*, using the differences of the X-Y pixel coordinates of their end points to index into a two-dimensional array of scaled angles. The angles for the two lines are then combined to determine the inside angle between them. For K=4, the scaled angle array used in the sequential implementation is (the scaling factor is Degrees/9, with angles oriented to correspond to the image coordinate system):

Table of Relative Angles

Distance in X

Distance in Y	-4	-3	-2	-1	0	1	2	3	4
-4	-15	-16	-17	-18	20	18	17	16	15
-3	-14	-15	-16	-18	20	18	16	15	14
-2	-13	-14	-15	-17	20	17	15	14	13
-1	-12	-12	-13	-15	20	15	13	12	12
0	-10	-10	-10	-10	0	10	10	10	10
1	- 8	- 8	- 7	- 5	0	5	7	8	8
2	- 7	- 6	- 5	- 3	0	3	5	6	7
3	- 6	- 5	- 4	- 2	0	2	4	5	6
4	- 5	- 4	- 3	- 2	0	2	3	4	5

The combining function is: $\theta = | \theta_1 - \theta_2 |$

If $\theta > 20$

Then $\theta = 40 - \theta$

Any point with $\theta = 10 \pm 2$ is flagged as a right angle (this flag will be used in the rectangle hypothesis generation step). An unnormalized one-dimensional Gaussian smoothing function is computed by forming the sum of the products of the angle values of the edge pixels within each 7-wide window (3 pixels in each direction from the center of the window) along the border. The window coefficients are:

3, 27, 90, 136, 90, 27, 3

The first derivative of curvature is computed by convolving the border with a mask of -1, 1 (i.e. computing the difference between each pixel on the border and one of its neighbors). Zero crossing points are computed by comparing the signs of neighboring border pixels. Corners are located by thresholding the smoothed curvature values and forming the logical AND of the resulting binary image with the binary image that results from the zero-crossing detection. (A curvature threshold of 1000 was used with the supplied test data)

9.2 Depth Image

9.2.1 Smoothing

Task: Smooth the depth image while preserving edges by applying a median filter with a 3×3 mask to the original depth data.

Recommended method: A new image is created where each pixel is the median value of the pixels in a 3×3 window centered on the same location in the original image. Pixels that are adjacent to the image boundary should simply retain the same values as in the original image.

9.2.2 Gradient Magnitude Computation

Task: Compute a standard 3×3 Sobel operation on the smoothed depth image. Pixels beyond the edge of the image should be treated as having values of zero. The gradient magnitude is the square root of the sum of the squares of the X and Y magnitudes that result from the Sobel.

Recommended Method: Any standard method is acceptable.

9.2.3 Threshold

Task: Select strong edge pixels by thresholding the output of the Sobel operation at a specified level. The result is a binary image where a 1 represents an edge pixel.

Recommended Method: Any standard method is acceptable. A threshold of 500 was used for the test data.

10. Intermediate-Level, High-Level, and Top-Down Processing

10.1 Rectangle Hypothesis Generation

Introduction: The goal of the initial rectangle hypothesis generation step is to find good candidate rectangles among the components of the intensity image so that model matching can begin. Note that the goal does not require every possible candidate rectangle to be found. Only some rectangles that can be reliably used in the matching process are of interest, because once an initial match is formed the remainder will be extracted by top-down processing of the depth data. This can be viewed as using strong cues to focus attention in the subsequent extraction of weaker or more ambiguous image events.

The result of rectangle hypothesis generation is a set of candidate rectangles. Each rectangle is described by six parameters: the coordinates of its center pixel, the lengths of its major and minor axes, the orientation of its major axis, and its intensity.

Task: Extract good quality rectangles with three or four corners visible in the intensity image. The input (for each connected component) is a list of the row and column coordinates of center points for corners that were detected on a component's boundary. Additionally, corners whose K-curvature was calculated to be close to 90 degrees are flagged. Corner points that are not on the convex hull of this point set are discarded. For each corner point on the hull, the angle formed by the computed lines that connect it to its neighboring corner points is computed. A component is declared to be a rectangle candidate if there are at least three contiguous right angle ($\pm$ two degrees) corners on the convex hull, which were also measured as right angles in terms of K-curvature. Two of the opposing corners are used to compute the center of the rectangle (which is taken to be the pixel nearest the midpoint of the diagonal line segment defined by the two corner points). The length of the major axis is the greatest distance between adjacent pairs of the rectangle's right angle corners. The orientation of the rectangle's major axis, relative to the origin, is also computed.

The length of the minor axis is the minimum distance between adjacent right-angle corners.

Recommended Method: If the K-curvature has not flagged at least 3 corners as right angles for a component, then reject the component as a rectangle. Otherwise the convex hull should be computed with a parallel Graham Scan type of algorithm. If there are less than 3 flagged corners on the hull, then the component is rejected as a rectangle. Next, corners of the hull that were flagged as right angles have their hull angles checked — i.e., angles formed between neighboring vertices of the hull are checked to determine whether or not they are approximately right angles. This is easily accomplished by exploiting the relationship between the vector dot product and the cosine of the angle as follows: as follows:

Given a triangle of points a,b,c with θ, the angle in question at b, then each point defines a corresponding vector A,B,C, from the origin to that point. Then we have

$$\cos\theta = \frac{(A-B)\cdot(C-B)}{|A-B|\ |C-B|}$$

squaring, we get

$$\cos^2\theta = \frac{((A-B)\cdot(C-B))^2}{((A-B)\cdot(A-B))((C-B)\cdot(C-B))}$$

Because there is an angular tolerance (E_θ) the final predicate becomes

$$\text{if } \cos^2\theta <= \cos^2(90 + E_\theta) \text{ AND } \cos^2\theta >= \cos^2(90 - E_\theta)$$

then right angle

else not right angle

This method avoids the use of inverse trigonometric and square-root operations. Note that the squares of the cosines for the range test can be assigned to global constants, since they are not affected by the particular set of points. If there are three successive right angles then the component is labeled a rectangle, otherwise it is rejected. Finally, if the component is a rectangle, its parameters are determined from the triangle defined by the three successive right angle corners as follows:

major-axis = length of the longer side

minor-axis = length of the shorter side

orientation = angle of the major axis with respect to the X-axis

center = pixel coordinates nearest the midpoint of the hypotenuse

10.2 Graph Matching

Introduction: The candidate rectangles can be thought of as forming a complete graph, where the links of the graph represent hypothetical mobile links between every pair of rectangles. Because the model is described by a graph structure, the matching process takes the form of finding the maximal set of constrained subgraph to subgraph isomorphisms between the complete graph formed by the candidates and the model graph. The matching process thus tries to match candidate rectangles to model rectangles by their size, intensity, and depth; and to match hypothetical links to model links by comparing the spatial relationships between rectangles that the links represent. Thus, to establish a subgraph isomorphism requires that a connected set of candidate rectangles and links be found that has the same properties and spatial relationships as some connected set of model rectangles and links.

Basically, the strategy that is followed is to find all of the single node isomorphisms (ie. matches between rectangles alone) and join them in pairs to form isomorphisms with a radius of one link, and then to merge those isomorphisms into larger subgraph isomorphisms. Note that initially there may be multiple matches to each node; however, as spatial relationships are used to further constrain the matches, these will be quickly reduced to the point that only a small number of possible matches remain. The test data set has been purposely designed so that the remaining ambiguous matches can only be fully resolved by analysis of the depth image. Once an initial set of matches has been found, the depth data is probed top-down to verify that the matched rectangles are at the proper depth as specified by the model. This step may eliminate some of the ambiguous matches, but not all.

After the graph-matcher confirms the depths of the rectangles that were found in

the intensity data, it continues trying to merge matched fragments to form larger subgraph isomorphisms. Whenever it is necessary to extract a rectangle from the data in order to extend a match, the graph-matcher probes the depth data in a top-down manner in order to locate a model rectangle that has not been found in the intensity image. The graph matcher also confirms the rectangle's depth and match strength using the routine described in section 10.3. Each rectangle that is established to exist in the depth data results in an update to the rectangle's position parameters in the model.

Task: Given the set of mobile models and the initial rectangle hypotheses from the intensity processing, form an initial subgraph to subgraph match of the rectangles and models. The initial match process starts by matching candidate and model rectangles (by size and intensity) and then tries to extend each match to a radius of one link (by comparing the spatial relationships of pairs of matched candidate rectangles to those of the corresponding pairs of adjacent model rectangles). Then confirm the model-specified depths of and intensities the matched rectangles by directed probing of the intensity and smoothed depth data (see section 10.3). Extend the matches for each model by model-directed, local probing of the thresholded gradient magnitude image for the presence of edges that correspond to each model rectangle (see section 10.4). The model-specified depth and intensity of each rectangle that is found by the probe is then verified by probing the intensity and smoothed depth images using the same technique as for confirming the depths of the initial rectangles (see sections 10.3 and 10.4.3).

The top-down probing of the depth data results in confirmation or rejection (veto) of the presence of each model rectangle in the depth data. Each confirmed rectangle is used to extend the model match, and update the positional information in the model. A rejected rectangle results in the rejection of whatever portion of a match depends on it. It is possible for an entire model to be rejected because one of its rectangles is rejected. For example, if previous processing has fixed the position of the model in the image so that there is only one possible pose for the rectangle, and

it is rejected by the depth probe, then there is no alternative but to reject the model. (Note that a rectangle is only rejected if there is strong evidence that it is not in the image. Occlusion does not cause a rejection).

Thus, in the process of extending a subgraph match, if it is necessary to link to a rectangle that is rejected, the entire subgraph will be labeled as rejected. Once a subgraph is rejected, no further attempt is made to extend it. However, it is possible that an attempt will be made to extend another subgraph to link with the rejected subgraph. If the other subgraph requires that link in order to be extended, then it too must be labeled as rejected. In this way, the rejection of a single rectangle can spread to whatever portion of a proposed match depends upon that rectangle.

Once verification has been completed for each model rectangle, and a global match has been computed for each model that is not rejected, an overall model match strength (the average of the node match strengths) is computed for each model. The model with the highest match strength is declared the best match and is used to produce the final output image.

Recommended Method: We begin with some definitions. In addition to the model graphs there are two globally maintained lists, called the probe list and the active root list. There are also two lists associated with each model rectangle, called the pose list and the compatible rectangle list.

Probe list: Contains a list of all model rectangle poses which may have empty link fields that await matching. The list is created as a result of the initial matching process and is used to guide the match extension process. The list is maintained in order according to the match strength of each rectangle pose. Thus, strong cues are given priority in the matching process. When a rectangle at the front of the probe list has all of its links filled in, it is removed from the probe list.

Active root list: Contains a list of all model rectangle poses that correspond to roots of models and that have not been eliminated through a veto. In other words, this is a list of all rectangles that are potential roots.

Compatible rectangle list: Each model rectangle has a list of the initially extracted rectangles that match its size and intensity. These lists are created from the set of initial candidate rectangles at the start of the initial graph-match process. Roughly square rectangles (those that still match the model rectangle after being rotated 90 degrees) are added to the list twice (once with the 90 degree rotation). Each rectangle that is added to the list is also duplicated with a 180 degree rotation to cover the symmetric orientation case.

Pose list: For each model rectangle a list of its possible poses in the scene is formed from the compatible rectangle list. The initial pose lists are created during the initial match phase and contain those compatible rectangles with at least one link to a compatible rectangle in a neighboring model rectangle's list of compatible rectangles. Thus, if two neighboring rectangles are linked, they will each appear in the pose list for their corresponding model rectangle.

The poses are the elements that will be linked together by the graph matcher as it tries to build isomorphic subgraphs. Poses either join an isomorphism to extend it, or are vetoed as a result of top-down probing of the depth and intensity data.

Given these definitions, the following five steps describe the initial matching process:

1. Build the compatible rectangle list for every model rectangle: Add to the compatible rectangle list of each model rectangle a copy of the parameters for each candidate rectangle whose size (within E_L) and intensity match the model rectangle. If a rectangle also matches the model after being rotated 90 degrees, then add a second copy to the list with that rotation. Lastly, add a second copy of each rectangle with a 180 degree rotation.

2. For each rectangle in the compatible rectangle list of each model rectangle, attempt to establish a link to every compatible rectangle associated with each neighboring model rectangle. A link is formed between two compatible rectangles if their spatial relationships (relative orientation, direction and distance) match the spatial relationship specified by the link between the corresponding model rectangles. Each successful

linkage is added to the pose lists of the model rectangles. If one compatible rectangle is linked to another that is already on a pose list, then the new link should be merged into the existing pose rather than becoming a new pose. By merging these poses, larger subgraph isomorphisms are created during the initial matching process, thus saving time in later stages. (Note that, initially, any symmetric pair of rectangles will be linked as two possible poses by virtue of the fact that the two rectangles are in the compatible rectangle lists of both model rectangles. A symmetric pair of rectangles would be two rectangles that have the same size and intensity and are oriented so that a 180 degree rotation of the pair is indistinguishable from the non-rotated pair.) The result of this step is the creation of pose lists, where each pose is an isomorphic subgraph that is created from the compatible rectangle lists. Many compatible rectangles will fail to link to any neighboring rectangles, and will be omitted from the pose lists. For this reason, the compatible rectangle lists will again be checked as part of the match extension process.

3. Form the initial, unordered probe list and active root list all of the rectangles in the pose lists.

4. For each rectangle on the probe list, probe the depth and intensity data for veto or match strength. If the rectangle is rejected its entire linked subgraph is removed from the probe and active root lists, and marked as rejected in the pose list and compatible rectangle list of the appropriate model rectangle. If the rectangle is not vetoed, a match strength is associated with it and it remains on the lists. When a rectangle has been probed, the corresponding rectangle in the compatible rectangle list is flagged as having been probed and is assigned the associated match strength.

5. The probe list is sorted by match strength so that the first element of the list has the greatest strength.

This concludes the initial match phase. Models that have completely empty pose lists will be rejected at this point. Note that the probe list may contain rectangles with very low match strength, because such rectangles are not necessarily vetoed. For example, a

rectangle may be mostly occluded, in which case it will have a low match strength but there is insufficient evidence to conclude that it is not present. Such a rectangle may be thought of as being hallucinated by the system. Because hallucinated rectangles are last on the probe list, they are only used to fill in missing links near the end of the match extension process.

The match extension process consists of the following four steps that are repeated until the probe list is empty.

6. The probe list is searched for the first empty link in a rectangle. Any rectangles at the front of the list that have all of their links filled are removed from the probe list. An empty link corresponds to a link in the model, and thus to a neighboring rectangle in the model. Note that all possible links between the initial set of rectangles have been found at this point. Thus, an empty link indicates a rectangle that must be added via the match extension process.

7. The object of this step is to make sure that there is a probed data rectangle corresponding to the selected neighboring rectangle. First, the neighbor's compatible rectangle list is searched. If a compatible rectangle is found that has been previously probed, then it should be used. If a compatible rectangle is found that hasn't been probed yet, then it must be probed. If no compatible rectangle is found, use the model parameters for the neighbor to direct a top-down search (local Hough transform) and then probe the area indicated by the search. Add the rectangle to the compatible rectangle list for the neighboring model rectangle, even if the search and/or the probe fails.

8. Once a probed data rectangle has been found for the neighbor, check the neighbor's pose list for any links to it. If a link is found, then the rectangle is part of a larger subgraph isomorphism which should be linked to the isomorphism that contains the pose at the top of the probe list. Otherwise, create a new pose consisting of just the newly probed rectangle and link the pose at the top of the probe list to it.

9. If the neighbor rectangle, or its instance on the pose list is flagged as vetoed, delete the

entire linked subgraph from the probe list (and the active root list, if necessary). Also, label as vetoed the entire structure associated with the model pose, corresponding to the probe rectangle. If it was not vetoed, and it was created and/or probed in step 7, then insert the new rectangle in the probe list according to its match strength and also add it to the active root if it corresponds to a model root.

Once the probe list is empty, the match extension process is complete. The result is a set of one or more model matches that are pointed to by the active root list.

10.3 Top-Down Probe for Confirmation or Veto of Initial Match

Introduction: For each rectangle that is found in the intensity data and matched in a model, the smoothed depth image and the original intensity image are probed to determine a match strength. The probe initially examines the area of the smoothed depth image that corresponds to the rectangle in its hypothesized pose. Pixels are counted that are found to be too distant with respect to the model-specified acceptable depth range. If the count exceeds a threshold then the match is vetoed. If the match is not vetoed, then an area of the intensity image is also probed that corresponds to the rectangle in its hypothesized pose. The result of the depth and intensity probes is a match strength indicated by the percentage of pixels in the area that are within the acceptable depth range and that have the correct intensity, minus the percentage of the depth pixels that are too deep.

Task: Given the parameters for a rectangle, probe the smoothed depth data and the intensity image to confirm that a region of roughly the correct depth and intensity is present. There are two parts to this task: a check for a strong indication that the rectangle is not present in the depth data, and the computation of a match strength.

For the first part of this task, a window of the smoothed depth image is selected that corresponds to the hypothesized pose of the rectangle. Within this area, the pixels are divided into three categories: those pixels that are deeper than the allowable range of depths for the rectangle, those that are within the allowable range, and

those that are too close. The number of too-deep pixels is determined, and if it exceeds a threshold, the rectangle is vetoed because it is either missing or not in the correct position.

The threshold is equal to

$$(E * E_{xy} + E_L) * (L + l)$$

where

L	=	Length of major axis
l	=	Length of minor axis
E_{xy}	=	Error in X and Y position
E_L	=	Error in axis lengths
E	=	Error multiplier for this match

(The value of E reflects the effects of previous processing on the potential error expressed by E_{xy}. In this case, because the position was actually computed from the intensity data, the positional accuracy is quite good and E_{xy} can be reduced. Thus E is equal to 0.25 for this match). In other words, the threshold is the maximum portion of the probe window in which too-deep pixels could appear because of the position and size errors that are created when the model is instantiated. Note that the above formula actually computes a slightly higher threshold than necessary in order to account for orientation error without performing additional computations involving angles.

If the rectangle is not vetoed, then a second step computes a match strength for the entire hypothesized rectangle pose area in the smoothed depth image and the intensity image. The match strength is reported as the percent of pixels that are both in the correct depth range and have the correct intensity minus the percentage of pixels that are too deep.

Recommended Method: In the smoothed depth image, within a window that corresponds to the hypothesized pose of the rectangle, the pixels are thresholded at

$D + E_D * D$. Pixels that exceed the threshold are too deep, and are counted. If their number exceeds $(E * E_{xy} + E_L) * (L + l)$, then the rectangle is vetoed.

If the rectangle is not vetoed, then a range test is performed on the smoothed depth image within the same window. The acceptable range is $D \pm E_D * D$, and pixels that pass the test form a binary image. Within a registered window of the intensity image, pixels are tested for intensity equal to I (the model rectangle's intensity). Another binary image results, which can then be combined via a logical AND with the other binary image to produce a third image representing those pixels that have the correct intensity and depth. The match strength for the rectangle is then computed by dividing the number of these correct pixels minus the number of too-deep pixels by the total number of pixels in the window area.

Note that if a rectangle is fully occluded, it will not be vetoed but it will have a match strength of zero or less.

10.4 Top-Down Examination of Depth Data

Introduction When it is necessary to search for a model rectangle in the depth data, a Hough transform is applied top-down within a spatially local window of the thresholded gradient magnitude image, followed by a search of the Hough array to locate evidence for parallel and perpendicular lines that form a rectangle with the appropriate size and orientation.

The rectangle location strategy is as follows: an X-Y oriented window that is large enough to contain the rectangle, given its possible position and size errors, is computed and used to mask the thresholded gradient magnitude image that resulted from the initial depth processing. Within that window, a Hough transform is computed in order to detect lines. The Hough array is then searched for the maximum bucket with approximately the same orientation as the major axis of the model rectangle. If a sufficiently strong bucket is found, the corresponding line is used to anchor the search for the other three edges of the rectangle. At least two of the remaining three edges must be detected as lines in the Hough array in order for a positive search

result to be returned. If only three of the edges are detected, the fourth is inferred from the model data.

10.4.1 Depth Window Size and Position

The rectangle search is given the position and size of an X-Y oriented search window and the expected orientation for the rectangle's major axis. The window's bounding edges are computed from the following formulas:

X	=	X position of rectangle's center
Y	=	Y position of rectangle's center
L	=	Length of major axis
l	=	Length of minor axis
θ	=	Orientation of major axis
E_{xy}	=	Error in X and Y position
E_L	=	Error in axis lengths
E_θ	=	Error in orientation
E	=	Error multiplier for this match
S	=	$sin(\theta)/2$
C	=	$\|cos(\theta)/2\|$

When the Hough transform is being used to locate a rectangle, the positional information that is used to place the window is based on the corresponding model rectangle. However, the estimated position of the model rectangle is relative to another rectangle (the rectangle at the top of the probe list) whose position has already been fixed in the image. Thus, the positional accuracy for the search depends on the accuracy of the position of the rectangle at the top of the probe list. If the match strength of that rectangle is high (≥ 0.5) then the positional accuracy is probably good and E is equal to one. If its match strength is low, then its positional accuracy is probably poor and so E is increased to two.

Window starting row=

$$Y - E*E_{xy} - S*(L + E_L) - C*(l + E_L)$$

(or 0 if result is < 0)

Window ending row=

$$Y + E*E_{xy} + S*(L + E_L) + C*(l + E_L)$$

(or 511 if result is > 511)

Window starting column=

$$X - E*E_{xy} - C*(L + E_L) - S*(l + E_L)$$

(or 0 if result is < 0)

Window ending column=

$$X + E*E_{xy} + C*(L + E_L) + S*(l + E_L)$$

(or 511 if result is > 511)

The rectangle locating routine searches within this window for a rectangle with its major axis oriented in the direction specified ($\theta \pm E*E_\theta$) or in the opposite direction ($(\theta + 180 \pm E*E_\theta) MOD\ 360$). The search consists of the steps in the following section.

10.4.2 Hough Transform and Rectangle Search

The Hough transform was chosen for this step for several reasons. The Hough transform tests some very interesting aspects of parallel architectures and is thus useful from a benchmarking point of view. Because it was previously implemented for the original benchmark, it should take less time to adapt it than would be required for implementing an entirely new operation. Also, several of the reviewers of the draft benchmark specifically requested that it be included. The constrained, verification mode of application of the Hough transform was arrived at after considerable experimentation. For example, initial attempts to apply the Hough transform to the entire image met with little success because the numerous strong straight edges combined to produce many false peaks and masked most of the true edges. Constraining the transform to a small window significantly improved the results, and further emphasized the top-down control aspect of this step. Also, the use of gradient orientation information from the Sobel was tried, but it was found that the results were

not sufficiently improved to warrant the additional complexity.

Task: Perform a model-directed search of the thresholded gradient magnitude image, in order to locate edges that correspond to a hypothesized pose of a model rectangle. The search begins with a Hough transform that is applied to the image within an X-Y oriented window that is sized and positioned so that it will contain the entire rectangle, even if it is subject to maximum errors in position, orientation and size.

The Hough array is then searched for strong lines that form a rectangle in the appropriate size and orientation ranges. If only three lines are found, a fourth is inferred from the model data. New position, size, and orientation values for the rectangle are computed from the extracted lines. If fewer than three lines are found, the original model data is returned unchanged.

Recommended Method:

1. Compute the Hough transform of the binary image (from step 8.2.3) within the specified window. The range of the Hough space is $0 \leq \theta \leq 360$, or $0 \leq rho \leq 724$, however the accumulator array has a resolution of 2° per bucket in θ, and 2 pixel widths per bucket in *rho*. Also note that, if desired, the window dimensions may be used to further constrain the range of *rho*, and the specified orientation range can be used to constrain the ranges of θ. Thus, it is possible for the size of the Hough array to be greatly reduced if that is desirable.

2. Search the Hough array to determine the maximum accumulator with θ in the allowable ranges for the rectangle's major axis orientation ($\theta \pm E * E_\theta$ and $(\theta + 180 \pm E * E_\theta) MOD 360$). The minimum allowable count in the selected accumulator is 30. If there is no accumulator with a count of at least 30, then the rectangle routine will report that no rectangle could be found.

3. If a major-axis line is found, the search proceeds by looking for a parallel line. Such a line must be at least $l - E_L$ distant from the first line, have a count of at least 30, and a θ within 3° of that of the first edge (ie. ± 1 bucket in θ).

4. The next step is to look for the strongest perpendicular line in the Hough array. The selected accumulator must have a count of at least 30 and be within 3° (± 1 θ bucket) of perpendicular to the first edge.

5. The last line must be found at least $L - E_L$ distant from the third line, have a count strength of at least 30, and be within 3° (± 1 θ bucket) of perpendicular to the first line.

6. If fewer than 3 lines were found, the rectangle routine reports that no rectangle is present in the window.

7. If 4 lines were found, the routine computes the new X and Y position of the center, the lengths of the major and minor axes, and returns these values together with the orientation of the major axis.

8. If 3 lines were found, the fourth line is inferred from the original expected dimensions of the rectangle. The new center position is computed by averaging two possible positions. The perpendicular bisector of the rectangle edge opposite to the missing line, is computed, and the point on the bisector that is closest to the original center position is taken as one of the possibilities. The other possibility is the bisector point that is $L/2$ (if the missing line is parallel to the minor axis) or $l/2$ (if the missing line is parallel to the major axis) distant from the base of the bisector in the direction of the rectangle's interior. The length of the axis perpendicular to the missing line is estimated by doubling the distance from the computed center position to the base of the perpendicular bisector. The routine returns the new center position, the estimated length of the axis perpendicular to the missing line, the computed length of the axis parallel to the missing line, and the orientation of the major axis.

10.4.3 Top-Down Probe for Confirmation or Veto

This is the same process that is used in section 10.3.1 for the initial depth confirmation. However, the rectangles that are being confirmed are those that were detected in the

depth data, rather than the intensity data. Because the rectangle parameters are identical in form, regardless of the source of the rectangle, the same task description and method as in section 10.3.1 apply here.

10.5 Result Presentation

Task: Create a display that indicates those model rectangles that were strongly matched and those that were not. The display will consist of the original intensity image with model rectangles brightly outlined. Rectangles whose match strength is equal to or exceeds the average match strength of the model will be outlined at intensity level 255. Rectangles with lower match strengths will be outlined at some arbitrary lower value that will be visually distinguished.

Recommended Method: The selected graph is traversed, and the updated positional information for each rectangle is used to draw its edges in the intensity image. Any order of traversal is acceptable, as is any standard method of drawing the lines in the image.

11. Required Timings and Instrumentation

Section to be Added

END

DATE

FILMED

5-88

DTIC